Claims

1. An interpreter performing operations specified in a computer program consisting of instructions, said instructions being translated to an intermediate format comprising an intermediate code for said instructions, using a service routine to perform the semantics of an instruction, comprising:

statistics means for collecting and recording statistics of how often service routines are executed and what parameters they had;

clustering means for grouping (SR89, SR17...SR6; SR4, SR34...SR16) frequently used service routines with program jumps between each other in a program function with regard to a predetermined frequency value for determining such service routines;

said statistics means recording the frequency of service routines executed after a said function; and

encoding means for assigning a frequently used service routines a shorter code than service routines (SR3, SR57, SR94...SR64) executed after a said function, thus gathered statistics, from an execution, control an encoding to optimize frequently used service routines for faster execution speed.

2. An interpreter according to claim 1, wherein more than one program function is grouped by grouping (SR89, SR17...SR6; SR4, SR34...SR16) frequently used service routines with jumps between each other in the same function, thus minimizing jumps between functions.

3. An interpreter according to claim 1, wherein said encoding means provides for a reduced bandwidth to an electronic memory when fetching intermediate code, as a frequently executed instruction is shorter.

4. An interpreter according to claim 1, wherein said statistics is collected before a simulator is compiled.

5. An interpreter according to claim 1, wherein said statistics is collected while a simulator is running, in which case the simulator dynamically updates a function with service

routines used to simulate an instruction set.

6. An interpreter according to claim 1, wherein the statistics and the encoding provides that a realistic instruction set which is translated to service routines can be written in a high-level programming language for a specific simulator task.

7. An interpreter according to claim 6, wherein the interpreter is provided to be written in the standard ISO C through said statistics and encoding.

8. An interpreter according to claim 1, wherein a program branch to a next service routine may be shared by all service routines in a function through said statistics and encoding.

9. An interpreter according to claim 8, wherein it reduces the number of jumps that are hard to predict in a branch prediction table, causing a table to functioning better on current processor architectures.

10. An interpreter according to claim 1, wherein profile driven compilation is used to further enhance a simulator performance through said statistics and encoding.

11. An interpreter according to claim 1, further providing for compiler register mapping of often used variables, as the variables may be allocated as local variables.

12. An interpreter according to claim 1, wherein it provides for avoiding use of compiler specific extensions, thus providing for compiler independence.

13. An interpreter according to claim 1, wherein it improves instruction cache performance by placing frequent code in a sequential block due to a common function for service routines.

14. An interpreter according to claim 1, wherein it is used by an emulator.

15. A method for an interpreter performing operations specified in a computer program consisting of instructions, said instructions being translated to an intermediate format comprising an intermediate code for said instructions, using a service routines to perform the semantics of an instruction, comprising the steps of:

collecting and recording statistics of how often service routines are executed and what parameters they had;

grouping (SR89, SR17...SR6; SR4, SR34...SR16) frequently used service routines with program jumps between each other in a program function with regard to a predetermined frequency value for determining such service routines;

said statistics recording the frequency of service routines executed after a said function; and

encoding for assigning frequently used service routines a shorter code than service routines (SR3, SR57, SR94...SR64) executed after a said function, thus gathered statistics, from an execution, control an encoding to optimize frequently used service routines for faster execution speed.

16. A method according to claim 15, wherein more than one program function is grouped by grouping (SR89, SR17...SR6; SR4, SR34...SR16) frequently used service routines with jumps between each other in the same function, thus minimizing jumps between functions.

17. A method according to claim 15, wherein said encoding provides for a reduced bandwidth to an electronic memory when fetching intermediate code, as a frequently executed instruction is shorter.

18. A method according to claim 15, wherein said statistics is collected before a simulator is compiled.

19. A method according to claim 15, wherein said statistics is collected while a simulator is running, in which case the simulator dynamically updates a function with service routines used to simulate an instruction set.

20. A method according to claim 15, wherein the statistics and the encoding provides that a realistic instruction set which is translated to service routines can be written in a high-level programming language for a specific simulator task.

21. A method according to claim 20, wherein the interpreter is provided to be written in the standard ISO C through said statistics and encoding.

22. A method according to claims 15, wherein a program branch to a next service routine may be shared by all service routines in a function through said statistics and encoding.

23. A method according to claim 22, wherein it reduces the number of jumps, which are hard to predict, in a branch prediction table, causing a table to functioning better on current processor architectures performance through said statistics and encoding.

24. A method according to claim 15, wherein profile driven compilation is used to further enhance a simulator performance through said statistics and encoding.

25. A method according to claim 15, wherein it provides for compiler register mapping of often used variables, as the variables may be allocated as local variables performance through said statistics and encoding.

26. A method according to claim 15, wherein it provides for avoiding use of compiler specific extensions, thus providing for compiler independence performance through said statistics and encoding.

27. A method according to claim 15, wherein it improves instruction cache performance by placing frequent code in a sequential block due to a common function for service routines.

28. A method according to claim 15, wherein it is used by an emulator.

29. An interpreter performing operations of computer program code instructions by means of service routines, comprising:

a statistics mechanism devised to register the frequency of execution and the execution parameters of the service routines;

a clustering mechanism devised to group frequently used service routines having mutual referring program jumps in a program function dependent on a predetermined frequency value.

30. The interpreter as recited in claim 29, wherein said statistics mechanism is devised to register the frequency of service routines executed after establishing said function.

31. The interpreter as recited in claim 29, further comprising an encoding mechanism devised to assign a frequently used service routine a shorter code than service routines executed after a said function.

32. A method for an interpreter performing operations of computer program code instructions by means of service routines, comprising the steps of:

registering the frequency of execution and the execution parameters of the service routines;

grouping frequently used service routines having mutual referring program jumps in a program function dependent on a predetermined frequency value.

33. The interpreter as recited in claim 32, further comprising the step of registering the frequency of service routines executed after establishing said function.

34. The interpreter as recited in claim 33, further comprising an encoding mechanism devised to assign a frequently used service routine a shorter code than service routines executed after a said function.

35. A computer program product comprising program code devised to direct a data processing system to perform the steps and functions of claims 15-34.